Package: mod (via r-universe)

September 1, 2024

Type Package

Title Lightweight and Self-Contained Modules for Code Organization

Version 0.1.4.9000

Description Creates modules inline or from a file. Modules can contain any R object and be nested. Each module have their own scope and package ``search path'' that does not interfere with one another or the user's working environment.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

URL `https://github.com/iqis/mod`

BugReports `https://github.com/iqis/mod/issues`

Suggests testthat (>= 2.1.0), covr, knitr, rmarkdown

VignetteBuilder knitr

Repository https://iqis.r-universe.dev

RemoteUrl https://github.com/iqis/mod

RemoteRef HEAD

RemoteSha a9daf6c64e546829b84e0983ef4eb74de10bacfb

Contents

as_module 2
drop 2
is_module 3
module 4
name 5
print.module 6
private 6
provide 7

refer 8
require 9
use 9

Index **11**

`as_module` *Use a Package as if a Module*

Description

Use a Package as if a Module

Usage

```
as_module(package)
```

Arguments

`package`	name of a package; character

Value

a `module` that contains a package's exported objects

Examples

```
tcltk <- as_module("tcltk")
ls(tcltk)

tcltk$is.tclObj(NULL)
```

`drop` *Detach a Module from the Search Path*

Description

If no argument is supplied, detach the most recently attached module.

Usage

```
drop(name)
```

Arguments

`name`	name of the module to exit from; character

Value

TRUE if successful; invisible

See Also

use

Examples

```
use(mod::ule({
   a <- 1
}), as = "my_module")

use(mod::ule({
   b <- 2
}), as = "my_other_module")

search()

# by name
drop("my_module")

# and at the head position
drop()

search()
```

---

is_module *Test if an Object is a Module*

Description

Test if an Object is a Module

Usage

```
is_module(x)
```

Arguments

x	An object

Value

TRUE if the object is a module, FALSE otherwise

module *Make a Module*

Description

Institute a module object inline or from a file. mod::ule() is a useful shorthand for module() when this package is not attached.

Usage

```
module(expr, parent = parent.frame(), lock = TRUE)

ule(expr, parent = parent.frame(), lock = TRUE)

acquire(path, parent = baseenv(), lock = TRUE)
```

Arguments

`expr`	module expression
`parent`	the enclosing environment
`lock`	lock the environment; logical
`path`	path to a module file

Details

Only use `lock = FALSE` for runtime debugging. It is otherwise necessary to keep the module locked.

Value

an `environment` of class `module` containing defined objects

See Also

`use`, `drop`

Examples

```
# from file
module_path <- system.file("misc", "example_module.R", package = "mod")
example_module <- acquire(module_path)

example_module$e(123)

# inline
my_module <- mod::ule({
    a <- 1
    .a <- 2
    f <- function(){.a}
})

my_module$a
my_module$f
```

name — *Name a Module*

Description

Name a Module

Usage

```
name(name)
```

Arguments

`name`	the name of the module; character

Value

the input

See Also

Other declaratives: `provide`, `refer`, `require`

Examples

```
mod::ule({
    name("my")
    # ...
})
```

print.module *Print a Module*

Description

Print a Module

Usage

```
## S3 method for class 'module'
print(x, ...)
```

Arguments

x	an object
...	dot-dot-dot, ignored

Value

the object itself; invisible

private *Extract the Private Environment of a Module*

Description

Extract the Private Environment of a Module

Usage

```
private(module)
```

Arguments

module	a module

Value

environment

Examples

```
m <- mod::ule({a <- 1})
pvt <- private(m)

ls(pvt, all.names = TRUE)
```

provide *Provide Objects from a Module*

Description

Can only be used inside a module expression. If this function is used, only the names included as argument are public. If not used, every name in the module will be public.

Usage

```
provide(...)
```

Arguments

`...`	name of any object to be accessible by user; name or character

Value

`NULL`; invisible

See Also

Other declaratives: `name`, `refer`, `require`

Examples

```
mod_a <- mod::ule({
    # names included in provide() are public, however...
    mod:::provide(var,.var, ..var)
    # It is suggested to omit mod::: when using
    var <- 1
    .var <- 2
    ..var <- 3 # objects denoted by .. prefix are always private.
    another_var <- 4 # objects not included in provide() are also private.
})

mod_b <- mod::ule({
    # if no call to provide(), all objects are public, except...
    var <- 1
    .var <- 2
    ..var <- 3 # objects denoted by .. prefix are always private.
})

ls(mod_a)
ls(mod_b)
```

refer *Copy Bindings from a Module to Another*

Description

Can only be used inside a module expression. Makes reference to objects from one module, with specified filters.

Usage

```
refer(..., include = c(), exclude = c(), prefix = "", sep = ".")
```

Arguments

`...`	names of modules; dot-dot-dot
`include`	names to include; character
`exclude`	names to excludde; character
`prefix`	prefix to names; character
`sep`	separator between prefix and names; character

Value

`NULL`; invisible

See Also

Other declaratives: `name`, `provide`, `require`

Examples

```
mod_a <- mod::ule(number <- 1)
mod_b <- mod::ule(number <- 2)

mod_c <- mod::ule({
    mod:::refer(mod_a, mod_b, prefix = .)
    # It is suggested to omit mod::: when using
    number <- mod_a.number + mod_b.number
})

mod_c$number
```

require — *Load/Attach Package to Local Search Path*

Description

Can only be used in a module expression. Emulates the effect of base::require() in its containing module, making functions and their chain of environment available. Will not automatically attach dependencies of the package, and the user must do it separately. Masks base::require() inside a module context. Unlike base::require(), gives an error when package is not installed.

Usage

```
require(package)
```

Arguments

`package`	name of the package; name or character

Value

`NULL`; invisible

See Also

Other declaratives: `name`, `provide`, `refer`

Examples

```
mod_tcl <- mod::ule({
    mod:::require(tcltk)
    # It is suggested to omit mod::: when using
    f <- tcl
})

identical(mod_tcl$f, tcltk::tcl)
```

use — *Attach a Module to the Search Path*

Description

If the module as a name, defined by name(), it will always be used for the search path.

Usage

```
use(module, as, parent = baseenv(), lock = TRUE)
```

Arguments

`module`	a module object, or path to a module file
`as`	name when attached to search; character
`parent`	the enclosing environment
`lock`	lock the environment; logical

Value

`TRUE` if successful; invisible

See Also

`drop`

Examples

```
module_path <- system.file("misc", "example_module.R", package = "mod")
example_module <- acquire(module_path)

# Attach module object to search path
use(example_module)
# or directly from file
use(module_path, "example_module")
```

Index

∗ **declaratives**
  name, 5
  provide, 7
  refer, 8
  require, 9

acquire *(module)*, 4
as_module, 2

drop, 2, *4*, *10*

is_module, 3

module, 4

name, 5, *7–9*

print.module, 6
private, 6
provide, *5*, 7, *8*, *9*

refer, *5*, *7*, 8, *9*
require, *5*, *7*, *8*, 9

ule *(module)*, 4
use, *3*, *4*, 9